



Soft Aspects

RIAs & Beyond



New light of Rich Internet Applications

Developing AJAX enabled File Browser

Table of Contents

| | |
|-------------------------------------|----|
| 1. Introduction..... | 3 |
| 2. Overview..... | 3 |
| 3. Step by step implementation..... | 4 |
| 4. Full sample source code..... | 11 |

1. Introduction

In this tutorial we'll guide you through step-by-step instructions of creating AJAX enabled browser for server-side files . The browser allows to show content of selected file , refresh files list and other useful operations that are executed via AJAX.

2. Overview

AJAX operations allow to send data to server without full page refresh. The components provide full featured support for AJAX operations that can be initiated by some events from the client side. For example clicking mouse on component, Drag and Drop operation, component refresh or other command from JavaScript, etc.

To enable AJAX operations it's required to set property **ajaxEnabled** = true.

In addition , the components support the following additional entities :

- **ajaxClientHandler** - user function to be called before sending AJAX request
- **ajaxListener** - server side listener to be called on the server side. This listener must implement **com.softaspects.faces.galileo.support.ajax.event.AjaxEventListener** interface.

If AJAX operation does not require components' refresh (like click on menu item) it is possible to set **ajaxTargetElement** property. This will allow to send response content to corresponding element on the client side (for example HTML div element). In addition , if property is not set, HTML div element will be created and added to the end of the document **automatically**. If AJAX operation requires to refresh the components (for example table refresh) this property will be ignored and the ones will be refreshed automatically.

3. Step by step implementation

We'll create managed bean file “`samples\ajax\FileViewerBean.java`” and JSP file “`fileViewer.jsp`”, but first of all we'll add managed bean to “`faces-config.xml`” file:

```
<managed-bean>
  <managed-bean-name>FileViewerBean</managed-bean-name>
  <managed-bean-class>samples.ajax.FileViewerBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

This will allow to use the managed bean as “`FileViewerBean`” JSF binding.

Now we'll create managed bean with dynamic tree data and selection models which will show file structure as tree. But before this we'll add some utility functions to create and fill tree data model:

```
/**
 * Creates tree data model from files structure
 *
 * @param rootFile file
 * @return tree data model
 */
public static BaseTreeDataModel getFilesTreeDataModel(File rootFile) {
    BaseTreeDataModel result = new BaseTreeDataModelImpl();
    result.addElement(getFileTreeItem(null, rootFile));
    return result;
}

/**
 * Recursive file to menu item convertor
 *
 * @param parentItem parent tree item
 * @param file current file
 * @return menu item
 */
public static TreeItem getFileTreeItem(TreeItem parentItem, File file) {
    if (file.isDirectory()) {
        TreeItem result = TreeRendererUtil.createFolder(parentItem,
            file.getAbsolutePath(), file.getName(),
            null, null, null, null);
        File[] files = file.listFiles();
        for (int i = 0; i < files.length; i++) {
            getFileTreeItem(result, files[i]);
        }
        return result;
    } else {
        TreeItem result = TreeRendererUtil.createDoc(parentItem,
            file.getName());
    }
}
```

```

        result.setData(file.getAbsolutePath());
        return result;
    }
}

```

“**FileViewerBean**” bean will contain all models for this sample. Now we’ll add data and selection models for the tree:

```

/**
 * Tree dynamic data model
 */
private BaseTreeDataModel dataModel = null;

/**
 * Tree selection model
 */
private BaseTreeSelectionModel selectionModel = null;

/**
 * Tree data model getter
 *
 * @return tree data model
 */
public BaseTreeDataModel getDataModel() {
    if (dataModel == null) {
        dataModel = getFilesTreeDataModel(
            new File(FacesUtil.getServletContext(
                FacesContext.getCurrentInstance()).getRealPath("/")));
    }
    return dataModel;
}

/**
 * Tree selection model getter
 *
 * @return tree selection model
 */
public BaseTreeSelectionModel getSelectionModel() {
    if (selectionModel == null) {
        selectionModel = new BaseTreeSelectionModelImpl();
        selectionModel.setCurrentSelection("0");
    }
    return selectionModel;
}

```

Now it’s time to create some JSP content to show the tree:

```

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://www.softaspects.com/wgf/tree" prefix="tree" %>
<%@ taglib uri="http://www.softaspects.com/wgf/flexMenu" prefix="menu" %>
<f:view>
<html>
<head>
    <title>File viewer sample</title>
<body>

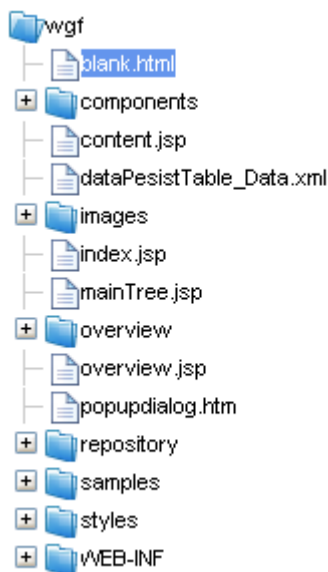
```

```

<h:form id="fileViewerSampleForm">
  <tree:tree id="tree1" varName="tree1"
    useServerSideLazyLoad="true">
    <tree:dataModel beanName="#{FileViewerBean.dataModel}"
      scope="session"/>
    <tree:selectionModel beanName="#{FileViewerBean.selectionModel}"
      scope="session"/>
  </tree:tree>
</h:form>
</body>
</html>
</f:view>

```

Finally , we've got the below output from our JSP page showing the files structure.



Let's add some functionality to show the content of selected item in the the tree . First of all, we'll setup the tree to enable AJAX mode and set the target element to a separate HTML DIV element (the response from the server will be sent to that DIV element). Also we'll add tree leaf selected listener allowing to show the file content in the DIV.

So, to enable AJAX mode we change tree definition in the JSP file, add **ajaxTarget** property and target HTML element:

```

<h:form id="fileViewerSampleForm">
  <h:panelGrid columns="2" rowClasses="top">
    <tree:tree id="tree1" varName="tree1" ajaxEnabled="true"
      treeLeafSelectedListener="#{FileViewerBean}"
      ajaxTargetElement="refreshDiv"
      useServerSideLazyLoad="true">
      ...
    </tree:tree>
  <f:verbatim>
    <div id="refreshDiv"></div>
  </f:verbatim>

```

```
</h:panelGrid>
</h:form>
```

Now we'll modify managed bean to implement **TreeLeafSelectedListener**:

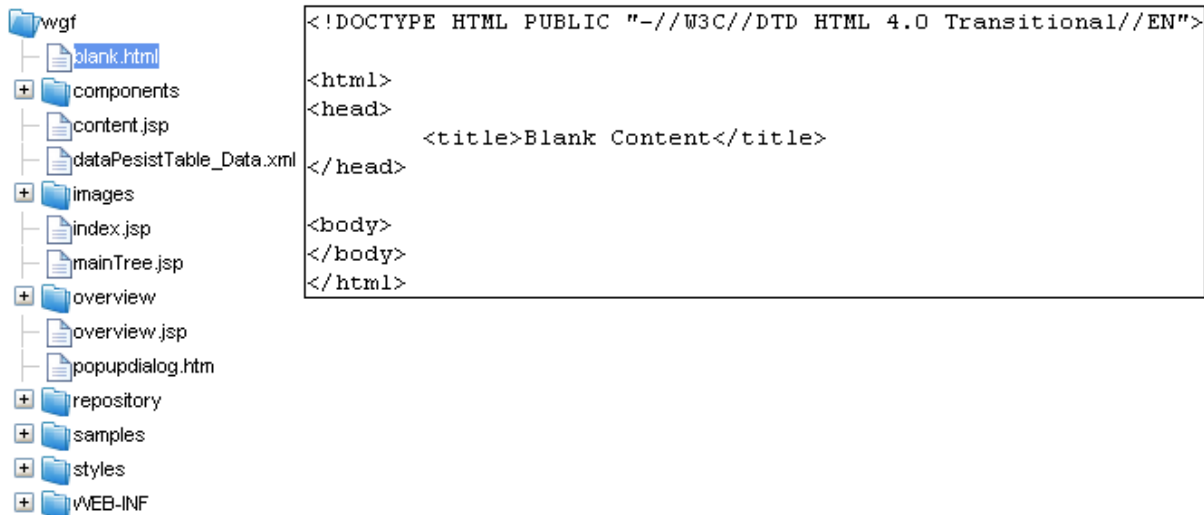
```
/**
 * Process tree item selection: return the selected file content to response
 *
 * @param event tree item selected event
 * @throws AbortProcessingException if exceptions
 */
public void processTreeLeafSelected(TreeLeafSelectedEvent event)
    throws AbortProcessingException {
    FacesContext facesContext = FacesContext.getCurrentInstance();
    if (facesContext != null) {
        try {
            facesContext.getResponseWriter().write(
                "<pre class=\"content\">");
            char[] buffer = new char[1028];
            HtmlUtils.writeText(facesContext.getResponseWriter(), buffer,
                getFileContent(getSelectedItemFile()));
            facesContext.getResponseWriter().write("</pre>");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

/**
 * Returns currently selected file or null if nothing is selected
 *
 * @return currently selected file or null if nothing is selected
 */
public String getSelectedItemFile() {
    TreeEntity item = getDataModel().getElementAt(
        "0," + getSelectionModel().getCurrentSelection());
    if (item instanceof TreeItem)
        return ((TreeItem) item).getData();
    return null;
}

/**
 * Read specified file to string
 *
 * @param file file to be read
 * @return file content
 * @throws IOException if I/O exceptions
 */
public String getFileContent(String file) throws IOException {
    StringBuffer result = new StringBuffer();
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(new FileInputStream(file)));
    String buffer;
    while ((buffer = reader.readLine()) != null) {
        result.append(buffer).append("\n");
    }
    return result.toString();
}
```

}

Finally , after all necessary changes , we can see the content of the selected file in the DIV element using AJAX .



Now we can add another AJAX component like menu to show how to execute server side operations. We'll be using FlexMenu component.

We use managed bean to provide dynamic menu data model:

```
/**
 * Flex menu for binding
 */
private FlexMenu menu;

/**
 * Flex menu items counter
 */
private int id = 0;
public static final String TEXT_FILE = "File";
private static final String TEXT_REFRESH = "Refresh";
private static final String TEXT_NEW = "New";
private static final String TEXT_OPEN = "Open...";
private static final String TEXT_PRINT = "Print...";
private static final String TEXT_CLOSE = "Close";
private static final String TEXT_VIEW = "View";
private static final String TEXT_COPY = "Copy";
private static final String TEXT_CUT = "Cut";
private static final String TEXT_PASTE = "Paste";
private static final String TEXT_HELP = "Help";
private static final String TEXT_ABOUT = "About";
```

```

/**
 * Flex menu binding setter
 *
 * @param menu flex menu
 */
public void setMenu(FlexMenu menu) {
    this.menu = menu;
}

/**
 * Flex menu getter
 *
 * @return flex menu
 */
public FlexMenu getMenu() {
    if (menu == null) {
        menu = new FlexMenu();
        FlexMenuConfiguration configuration =
            new FlexMenuConfiguration();
        configuration.setSubMenuIndicatorImageSrc(
            "{pathToImages}flexMenu/submenuIcon.gif");
        menu.setMenuConfiguration(configuration);
        FlexMenuItem fileItem = getMenuItems(menu, TEXT_FILE);
        getMenuItems(fileItem, TEXT_REFRESH);
        getMenuItems(fileItem, TEXT_NEW);
        getMenuItems(fileItem, TEXT_OPEN);
        getMenuItems(fileItem, TEXT_PRINT);
        getMenuItems(fileItem, TEXT_CLOSE);
        FlexMenuItem viewItem = getMenuItems(menu, TEXT_VIEW);
        getMenuItems(viewItem, TEXT_COPY);
        getMenuItems(viewItem, TEXT_CUT);
        getMenuItems(viewItem, TEXT_PASTE);
        FlexMenuItem helpItem = getMenuItems(menu, TEXT_HELP);
        getMenuItems(helpItem, TEXT_HELP);
        getMenuItems(helpItem, TEXT_ABOUT);
    }
    return menu;
}

/**
 * Flex menu items creator function
 *
 * @param parent parent item
 * @param title item title
 * @return flex menu item
 */
private FlexMenuItem getMenuItems(UIComponent parent, String title) {
    FlexMenuItem result = new FlexMenuItem();
    result.setId("menuId" + id++);
    result.setText(title);
    result.setRendered(true);
    result.setEnabled(true);
    if (parent != null)
        parent.getChildren().add(result);
    return result;
}

```

In JSP file we add one line with the component definition and the binding:

```
<menu:flexMenu id="menu" varName="vMenu" binding="#{FileViewerBean.menu}"
  ajaxListener="#{FileViewerBean}" ajaxEnabled="true"/>
```

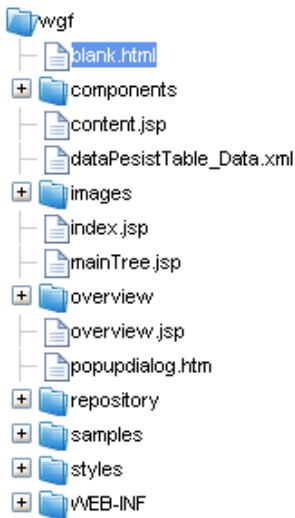
We can skip setting **ajaxTargetElement** property - in this case the target element will be created automatically, but setting **ajaxEnable** property is required to allow **AJAX** mode. Also we'll use **ajaxListener** property to handle menu item selection on the server side, that's why we have to implement **AjaxEventListener** and make some changes to the managed bean:

```
public void processAjaxEvent(AjaxEvent event)
    throws AbortProcessingException {
    if (menu != null) {
        FlexMenuItem menuItem = menu.getSelectedItem();
        FacesContext facesContext = FacesContext.getCurrentInstance();
        if ((facesContext != null) && (menuItem instanceof FlexMenuTextItem)) {
            String text = ((FlexMenuTextItem) menuItem).getText();
            StringBuffer result = new StringBuffer();
            result.append(RenderingUtils.createJScriptBeginTag());
            if (text.equals(TEXT_REFRESH)) {
                dataModel = null;
                UIComponent component = facesContext.getViewRoot().findComponent(
                    "fileViewerSampleForm:tree1");
                if (component instanceof FacesTree)
                    ((FacesTree) component).setDataModel(getDataModel());
                result.append("tree1.refresh();");
            } else if (text.equals(TEXT_ABOUT)) {
                result.append(AjaxUtils.getAlertFunction(
                    "File viewer sample v. 1.0"));
            }
            result.append(RenderingUtils.createJScriptEndTag());
            try {
                facesContext.getResponseWriter().write(result.toString());
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

In that handler we get the selected menu item and make some operations on the server side : tree data model changing and sending tree refresh JavaScript command in the response .

Once all the above steps implemented , we're able to run the application browsing the content of the server-side files using AJAX .

File View Help



```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
    <title>Blank Content</title>
</head>
<body>
</body>
</html>

```

4. Full sample source code

File fileViewer.jsp

```

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://www.softaspects.com/wgf/tree" prefix="tree" %>
<%@ taglib uri="http://www.softaspects.com/wgf/flexMenu" prefix="menu" %>
<f:view>
  <html>
  <head>
    <title>File viewer sample</title>
    <style type="text/css">
      .top {
        vertical-align: top;
      }

      .content {
        border: 1px solid black;
      }
    </style>
  <body>
    <h:form id="fileViewerSampleForm">
      <menu:flexMenu id="menu" varName="vMenu" binding="#{FileViewerBean.menu}"
        ajaxListener="#{FileViewerBean}" ajaxEnabled="true"/>
      <h:panelGrid columns="2" rowClasses="top">
        <tree:tree id="tree1" varName="tree1" ajaxEnabled="true"
          treeLeafSelectedListener="#{FileViewerBean}"
          ajaxTargetElement="refreshDiv" useServerSideLazyLoad="true">
          <tree:dataModel beanName="#{FileViewerBean.dataModel}"

```

```

        scope="session"/>
        <tree:selectionModel beanName="#{FileViewerBean.selectionModel}"
        scope="session"/>
    </tree:tree>
    <f:verbatim>
        <div id="refreshDiv"></div>
    </f:verbatim>
</h:panelGrid>
</h:form>
</body>
</html>
</f:view>

```

File FileViewerBean.java

```

package samples.ajax;

import com.softaspects.faces.galileo.components.flexmenu.FlexMenu;
import com.softaspects.faces.galileo.components.flexmenu.FlexMenuItem;
import com.softaspects.faces.galileo.components.flexmenu.FlexMenuItemTextItem;
import
com.softaspects.faces.galileo.components.flexmenu.configuration.FlexMenuConfigur
ation;
import com.softaspects.faces.galileo.components.tree.FacesTree;
import com.softaspects.faces.galileo.events.tree.TreeLeafSelectedEvent;
import com.softaspects.faces.galileo.listeners.tree.TreeLeafSelectedListener;
import com.softaspects.faces.galileo.support.FacesUtil;
import com.softaspects.faces.galileo.support.ajax.AjaxUtils;
import com.softaspects.faces.galileo.support.ajax.event.AjaxEvent;
import com.softaspects.faces.galileo.support.ajax.event.AjaxEventListener;
import com.softaspects.framework.galileo.components.tree.TreeItem;
import com.softaspects.framework.galileo.components.treemodel.*;
import com.softaspects.framework.galileo.renderers.html.tree.TreeRendererUtil;
import com.softaspects.framework.galileo.support.renderers.RenderingUtils;
import com.sun.faces.util.HtmlUtils;

import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import java.io.*;

/**
 * Bean for file view sample
 */
public class FileViewerBean implements
    TreeLeafSelectedListener, AjaxEventListener {

    /**
     * Tree dynamic data model
     */
    private BaseTreeDataModel dataModel = null;

    /**
     * Tree selection model
     */
    private BaseTreeSelectionModel selectionModel = null;

```

```

/**
 * Flex menu for binding
 */
private FlexMenu menu;

/**
 * Flex menu items conunter
 */
private int id = 0;
public static final String TEXT_FILE = "File";
private static final String TEXT_REFRESH = "Refresh";
private static final String TEXT_NEW = "New";
private static final String TEXT_OPEN = "Open...";
private static final String TEXT_PRINT = "Print...";
private static final String TEXT_CLOSE = "Close";
private static final String TEXT_VIEW = "View";
private static final String TEXT_COPY = "Copy";
private static final String TEXT_CUT = "Cut";
private static final String TEXT_PASTE = "Paste";
private static final String TEXT_HELP = "Help";
private static final String TEXT_ABOUT = "About";

/**
 * Flex menu binding setter
 *
 * @param menu flex menu
 */
public void setMenu(FlexMenu menu) {
    this.menu = menu;
}

/**
 * Flex menu getter
 *
 * @return flex menu
 */
public FlexMenu getMenu() {
    if (menu == null) {
        menu = new FlexMenu();
        FlexMenuConfiguration configuration = new FlexMenuConfiguration();
        configuration.setSubMenuIndicatorImageSrc(
            "{pathToImages}flexMenu/submenuIcon.gif");
        menu.setMenuConfiguration(configuration);
        FlexMenuBaseItem fileItem = getMenuItem(menu, TEXT_FILE);
        getMenuItem(fileItem, TEXT_REFRESH);
        getMenuItem(fileItem, TEXT_NEW);
        getMenuItem(fileItem, TEXT_OPEN);
        getMenuItem(fileItem, TEXT_PRINT);
        getMenuItem(fileItem, TEXT_CLOSE);
        FlexMenuBaseItem viewItem = getMenuItem(menu, TEXT_VIEW);
        getMenuItem(viewItem, TEXT_COPY);
        getMenuItem(viewItem, TEXT_CUT);
        getMenuItem(viewItem, TEXT_PASTE);
        FlexMenuBaseItem helpItem = getMenuItem(menu, TEXT_HELP);
        getMenuItem(helpItem, TEXT_HELP);
        getMenuItem(helpItem, TEXT_ABOUT);
    }
    return menu;
}

```

```

}

/**
 * Flex menu items creator function
 *
 * @param parent parent item
 * @param title item title
 * @return flex menu item
 */
private FlexMenuItem getMenuItem(UIComponent parent, String title) {
    FlexMenuItem result = new FlexMenuItem();
    result.setId("menuId" + id++);
    result.setText(title);
    result.setRendered(true);
    result.setEnabled(true);
    if (parent != null)
        parent.getChildren().add(result);
    return result;
}

/**
 * Tree data model getter
 *
 * @return tree data model
 */
public BaseTreeDataModel getDataModel() {
    if (dataModel == null) {
        dataModel = getFilesTreeDataModel(new File(FacesUtil.getServletContext(
            FacesContext.getCurrentInstance()).getRealPath("/")));
    }
    return dataModel;
}

/**
 * Tree selection model getter
 *
 * @return tree selection model
 */
public BaseTreeSelectionModel getSelectionModel() {
    if (selectionModel == null) {
        selectionModel = new BaseTreeSelectionModelImpl();
        selectionModel.setCurrentSelection("0");
    }
    return selectionModel;
}

/**
 * Creates tree data model from files structure
 *
 * @param rootFile file
 * @return tree data model
 */
public static BaseTreeDataModel getFilesTreeDataModel(File rootFile) {
    BaseTreeDataModel result = new BaseTreeDataModelImpl();
    result.addElement(getFileTreeItem(null, rootFile));
    return result;
}

```

```

/**
 * Recursive file to menu item convertor
 *
 * @param parentItem parent tree item
 * @param file current file
 * @return menu item
 */
public static TreeItem getFileTreeItem(TreeItem parentItem, File file) {
    if (file.isDirectory()) {
        TreeItem result = TreeRendererUtil.createFolder(
            parentItem, file.getAbsolutePath(), file.getName(),
            null, null, null, null);
        File[] files = file.listFiles();
        for (int i = 0; i < files.length; i++) {
            getFileTreeItem(result, files[i]);
        }
        return result;
    } else {
        TreeItem result = TreeRendererUtil.createDoc(
            parentItem, file.getName());
        result.setData(file.getAbsolutePath());
        return result;
    }
}

/**
 * Returns currently selected file or null if nothing is selected
 *
 * @return currently selected file or null if nothing is selected
 */
public String getSelectedItemFile() {
    TreeEntity item = getDataModel().getElementAt(
        "0," + getSelectionModel().getCurrentSelection());
    if (item instanceof TreeItem)
        return ((TreeItem) item).getData();
    return null;
}

/**
 * Read specified file to string
 *
 * @param file file to be read
 * @return file content
 * @throws IOException if I/O exceptions
 */
public String getFileContent(String file) throws IOException {
    StringBuffer result = new StringBuffer();
    BufferedReader reader = new BufferedReader(new InputStreamReader(
        new FileInputStream(file)));
    String buffer;
    while ((buffer = reader.readLine()) != null) {
        result.append(buffer).append("\n");
    }
    return result.toString();
}

/**
 * Process tree item selection: return to response selected file content

```

```

*
* @param event tree item selected event
* @throws AbortProcessingException if exceptions
*/
public void processTreeLeafSelected(TreeLeafSelectedEvent event)
    throws AbortProcessingException {
    FacesContext facesContext = FacesContext.getCurrentInstance();
    if (facesContext != null) {
        try {
            facesContext.getResponseWriter().write("<pre class=\"content\">");
            char[] buffer = new char[1028];
            HtmlUtils.writeText(facesContext.getResponseWriter(), buffer,
                getFileContent(getSelectedItemFile()));
            facesContext.getResponseWriter().write("</pre>");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

/**
* Ajax event handler
* @param event ajax event
* @throws AbortProcessingException
*/
public void processAjaxEvent(AjaxEvent event)
    throws AbortProcessingException {
    if (menu != null) {
        FlexMenuItem menuItem = menu.getSelectedItem();
        FacesContext facesContext = FacesContext.getCurrentInstance();
        if ((facesContext != null) && (menuItem instanceof FlexMenuTextItem)) {
            String text = ((FlexMenuTextItem) menuItem).getText();
            StringBuffer result = new StringBuffer();
            result.append(RenderingUtils.createJScriptBeginTag());
            if (text.equals(TEXT_REFRESH)) {
                dataModel = null;
                UIComponent component = facesContext.getViewRoot().findComponent(
                    "fileViewerSampleForm:tree1");
                if (component instanceof FacesTree)
                    ((FacesTree) component).setDataModel(getDataModel());
                result.append("tree1.refresh();");
            } else if (text.equals(TEXT_ABOUT)) {
                result.append(AjaxUtils.getAlertFunction(
                    "File viewer sample v. 1.0"));
            }
            result.append(RenderingUtils.createJScriptEndTag());
            try {
                facesContext.getResponseWriter().write(result.toString());
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
}

```